```
pragma solidity ^0.5.3;

interface IERC20 {

//totalSupply - it returns the initial quantity of rolled out tokens
    function totalSupply() external view returns (uint256);

//balanceOf - it returns the number of token hold by any particular
address
    function balanceOf(address _owner) external view returns (uint256
balance);

//transfer -  it is to trnasfer the token from one account to other
account
    function transfer(address _to, uint256 _value) external returns
(bool success);

//approve - owner approves a spender to use it's own token

    function approve(address _spender, uint256 _value) external
returns (bool success);

//transferFrom - once approved, it is used to transfer all or partial
allowed/approved
//tokens to spender
    function transferFrom(address _from, address _to, uint256 _value)
external returns (bool success);

//allowance -  it is to know the number of remaining approved tokens
    function allowance(address _owner, address _spender) external view
returns (uint256 remaining);

//trnasfer event - it  is used to log the transfer function activity
like from account, to account
//and how much token was transfered
    event Transfer(address indexed _from, address indexed _to, uint256
_value);

//approval event  - it is used inside approved function to log the
activity of approved function
    event Approval(address indexed _owner, address indexed _spender,
uint256 _value);

}

contract MyERC20Token is IERC20{

    mapping (address => uint256) public _balances;

    //Approval
    mapping (address => mapping(address => uint256)) _allowed;
    //1111 => 2222 - 10
    //1111 => 3333 - 20

    //name , symbol, decimal

    string public name = "Blockstraining";
    string public symbol = "BLKTRN1";
    uint public decimals = 0;

    //uint256 - intial supply
    uint256 private _totalSupply;

    //address - creator's address
    address public _creator;

    constructor() public{
        _creator = msg.sender;
        _totalSupply = 50000;
        _balances[_creator] = _totalSupply;
    }

    function totalSupply() external view returns (uint256){
        return _totalSupply;
    }

    function balanceOf(address _owner) external view returns (uint256
balance){
        return _balances[_owner];
    }

    function transfer(address _to, uint256 _value) public returns
(bool success){
        require(_value > 0 && _balances[msg.sender] >= _value);

        _balances[_to] += _value;
        _balances[msg.sender] -=_value;

        emit Transfer(msg.sender, _to, _value);

        return true;


    }

    function approve(address _spender, uint256 _value) public returns
(bool success){
        require(_value > 0 && _balances[msg.sender] >= _value);

        _allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender,  _spender, _value);

        return true;

    }

     function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success){
        require(_value > 0 && _balances[_from] >= _value &&
_allowed[_from][_to] >= _value);

        _balances[_to] += _value;
        _balances[_from] -=_value;

        _allowed[_from][_to] -= _value;

        return true;

     }

     function allowance(address _owner, address _spender) external
view returns (uint256 remaining){
         return _allowed[_owner][_spender];
     }
}

contract ICOBLK is MyERC20Token{

    //define the admin of ICO
    address public administrator;

    //Recipient account
    address payable public recipient;

    //set price of token , 0.001 ether = 1000000000000000
    uint public tokenPrice = 1000000000000000;

    //hardcap 500 ether = 500000000000000000000
    uint public icoTarget = 500000000000000000000;

    //define a state variable to track the funded amount
    uint public receivedFund;

    //maximum (10 ether) & minimum(0.001) investment allowed

    uint public maxInvestment = 10000000000000000000;
    uint public minInvestment = 1000000000000000;

    //set the ICO status
    enum Status {inactive, active, stopped, completed}

    Status public icoStatus;

    uint public icoStartTime = now;

    //5 days - duration
    uint public icoEndTime = now + 432000;

    //trading start time
    //uint public startTrading = icoEndTime + 432000;

    //uint public startTrading = icoEndTime;
    uint public startTrading = now;

    modifier ownerOnly{
        if(msg.sender == administrator){
            _;
        }
    }

    constructor (address payable _recipient) public{
        administrator = msg.sender;
        recipient = _recipient;
    }

    function setStopStatus() public ownerOnly{
        icoStatus = Status.stopped;
    }

    function setActiveStatus() public ownerOnly{
        icoStatus = Status.active;
    }

    function getIcoStatus() public view returns(Status){
        if (icoStatus == Status.stopped){
            return Status.stopped;
        }else if (block.timestamp >=icoStartTime && block.timestamp
<=icoEndTime){
            return Status.active;
        }else if (block.timestamp <= icoStartTime){
            return Status.inactive;
        }else{
            return Status.completed;
        }
    }


    function Investing() payable public returns(bool){
        //check for hard cap
        icoStatus = getIcoStatus();

        require(icoStatus == Status.active, "ICO in not active");

        require(icoTarget >= receivedFund + msg.value, "Target
Achieved. Investment not accepted");

        //check for minimum and maximum investment
        require(msg.value >= minInvestment && msg.value <=
maxInvestment, "Investment not in allowed range");

        uint tokens = msg.value / tokenPrice;

        _balances[msg.sender] +=tokens;

        _balances[_creator] -=tokens;

        recipient.transfer(msg.value);

        receivedFund += msg.value;

        return true;
    }

    function burn() public ownerOnly returns(bool){
        icoStatus = getIcoStatus();

        require( icoStatus == Status.completed, "ICO not complete");

        _balances[_creator] = 0;
    }

    function transfer(address _to, uint256 _value) public returns
(bool success){
        require(block.timestamp > startTrading, "Trading is not
allowed currently");
        super.transfer(_to, _value);
        return true;

    }

    function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success){
        require(block.timestamp > startTrading, "Trading is not
allowed currently");
        super.transferFrom(_from, _to, _value);
        return true;
    }
}
```